Anime List Design Doc

1. Preamble

This document will provide an overview of the system architecture for our application.

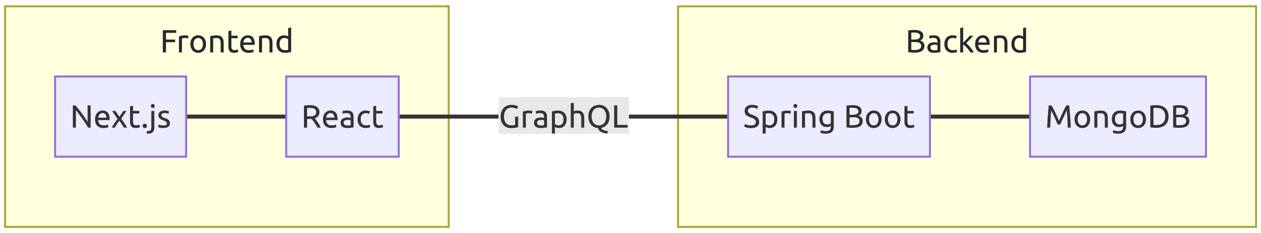
Although this document is formatted as Markdown, it contains diagrams that will not be rendered by GitHub. So we will maintain a PDF copy of this document here.

2. Background

Our application provides a platform for anime viewers who are interested in maintaining a collection of the series they've watched and are currently watching. Some of the basic features for this type of application include adding entries to a list, setting a status (e.g. plan to watch, currently watching, watched) for each one, and adding a rating for each one.

There are already some capable sites that fulfill this purpose, such as MyAnimeList (MAL) and Anilist. We plan to differentiate ourselves from these existing offerings by catering to a "power-user" group of users that would benefit from a greater degree of customizability. Before apps like MyAnimeList, using a spreadsheet was a common solution for this use case, which required significant manual work but allowed complete customization to the user's needs. We aim to have our app closer to spreadsheets on the customization spectrum, without sacrificing much convenience.

3. System Architecture



The above diagram shows a high-level diagram of our system architecture.

Our backend consists of two parts: a database for persistence and the main server that handles requests.

For our database, we use MongoDB Atlas. This provides us with a flexible NoSQL database where we can store documents in a JSON format. One primary benefit of this is the ability to model our data within the database in a form that closely matches the way we model the received data in the frontend. Rather than having JSON-shaped data on the frontend and tabular data in the backend, both of them can share a similar structure.

For the main backend server, we use the Spring Boot framework for Java. This allows us to hook into the robust Java ecosystem, giving us libraries to interact with the MongoDB database as well as handle GraphQL requests, which will be discussed a few paragraphs down. In our production deployment, we host this backend server on Heroku.

Our frontend also consists of two parts: the Next.js server that serves the frontend files, and the actual frontend files themselves.

The Next.js server is somewhat of a black box since the implementation is part of the Next.js framework. However, it is still a distinct module in our system architecture, so we will discuss it here. This server is responsible for responding to the client's browser requests and serving the appropriate React frontend files. For example, when the user navigates to `/` the server will respond with the frontend files corresponding to the index page. In our production deployment, this server is hosted on Vercel.

Second, the frontend files that actually run in the user's browser are implemented using React. These files make up the "client" portion of our application, and are responsible for communicating with the backend and displaying the UI of our app.

Finally, as previously mentioned, we use GraphQL to communicate between the frontend and backend! GraphQL is an API specification that acts as an alternative to traditional REST APIs. Rather than making a request to a REST endpoint and receiving data back with a shape determined by the endpoint, GraphQL allows us to traverse a graph of data and specify exactly what objects and fields we want to retrieve. It also allows us to get all of the data we need in a single request, rather than having to make multiple requests to retrieve all of the necessary data. This is because the data is represented as nodes and fields, where the fields can themselves be nodes.

4. UX Considerations

As briefly mentioned above, we want our app to be highly customizable and personal compared to the experiences offered by existing anime list apps. However, we recognize that a high degree of customizability can sometimes come at the cost of convenience, which is something we want to avoid. Throughout the design process, we have aimed to balance these two factors.

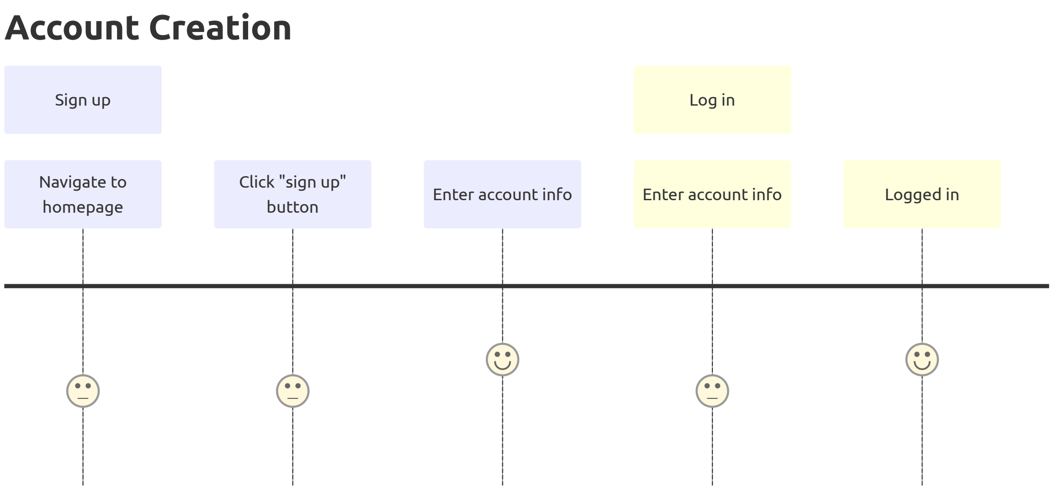
Furthermore, an important step in our design and implementation was to consider minor details in the interface and user flow, which we consider essential to creating an exceptional user experience.

The following bullet list outlines the user flow that we envisioned:

- User visits site for the first time and sees landing page.
- User creates an account for the site, then logs in.
- (Optional) User configures custom rating systems for their lists.
- User creates anime lists using preset or custom rating systems.
- [Loop] User adds entries to their list via search.
- User edits the watch status and ratings for their entries.
- User customizes their profile and shows the lists they've curated.
- Go back to the [Loop] step. User continues to use the app as they continue to watch anime.

The following subsections will show some high-level user flows for common processes in our app.

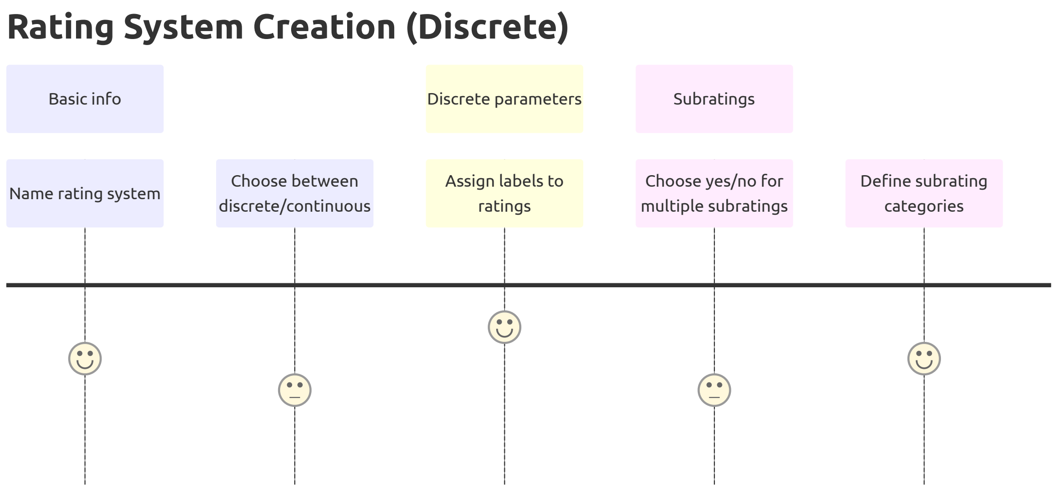
4.1. Account Creation

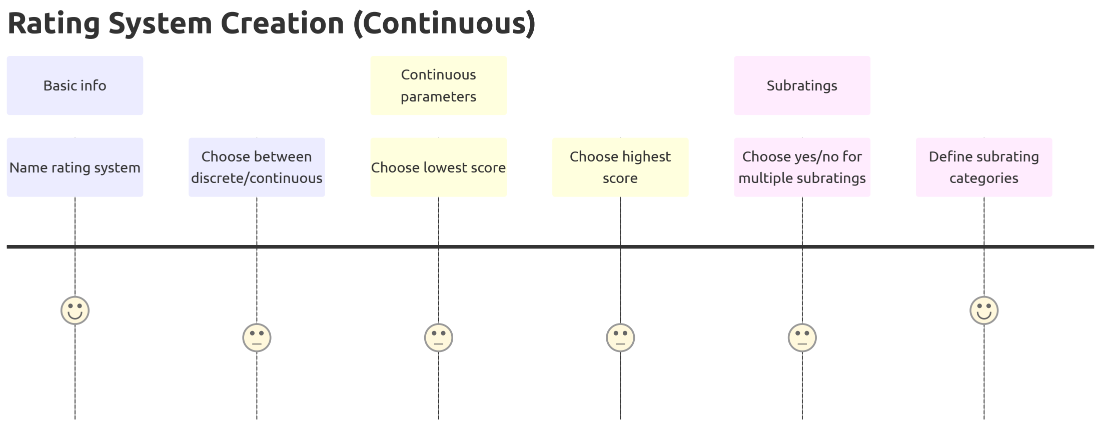


4.2. Rating System Creation

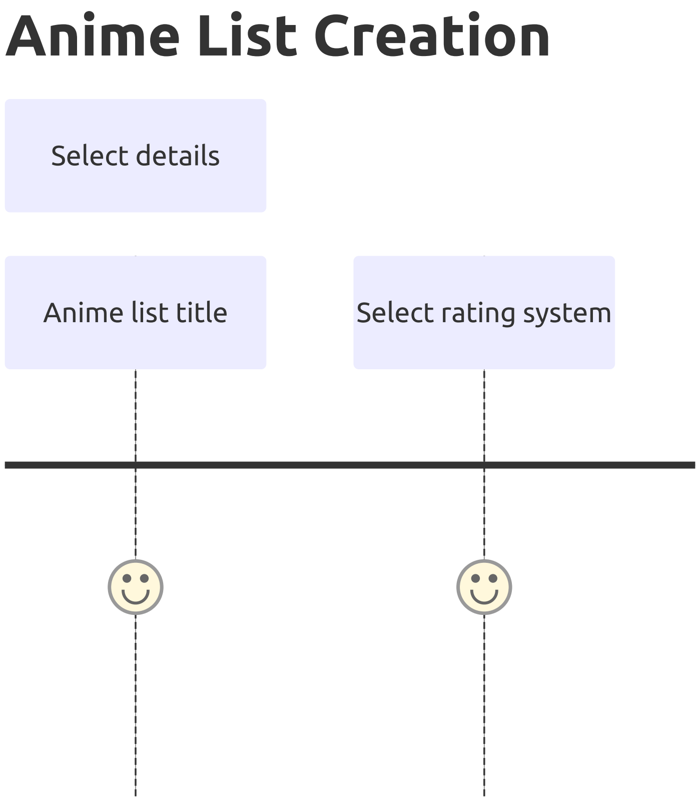
Note: we have a separate doc detailing why this system exists and how it works.

For now, the gist of it is that users can create custom rating systems to associate with their lists. Each rating system can either be discrete (discrete scores can be assigned) or continuous (scores can be any real number in a given range).

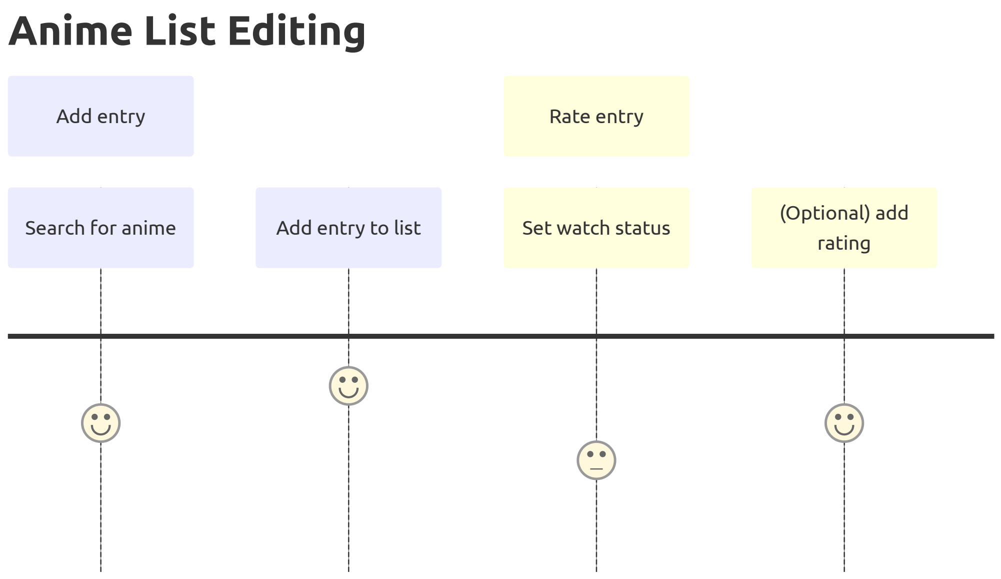

Rating System Creation (Discrete)
Basic info
Discrete parameters
Subratings
Name rating system
Choose between discrete/continuous
Assign labels to ratings
Choose yes/no for multiple subratings
Define subrating categories


Rating System Creation (Continuous)
Basic info
Continuous parameters
Subratings
Name rating system
Choose between discrete/continuous
Choose lowest score
Choose highest score
Choose yes/no for multiple subratings
Define subrating categories

4.3. Anime List Creation

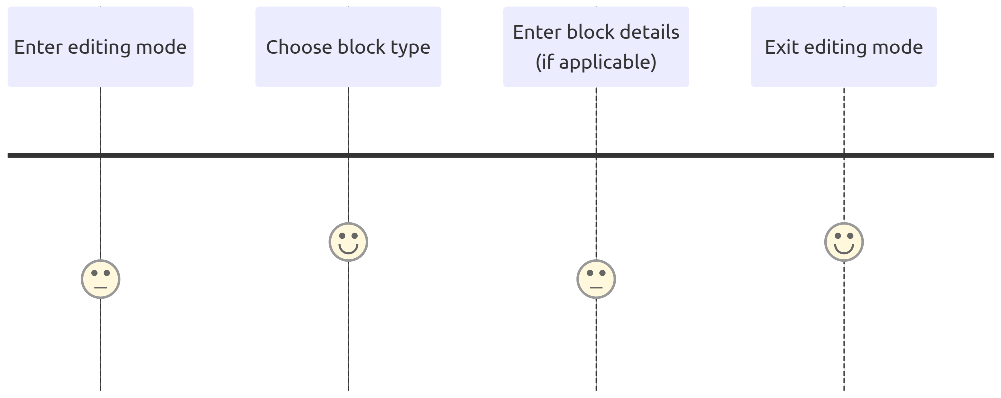


4.4. Anime List Editing

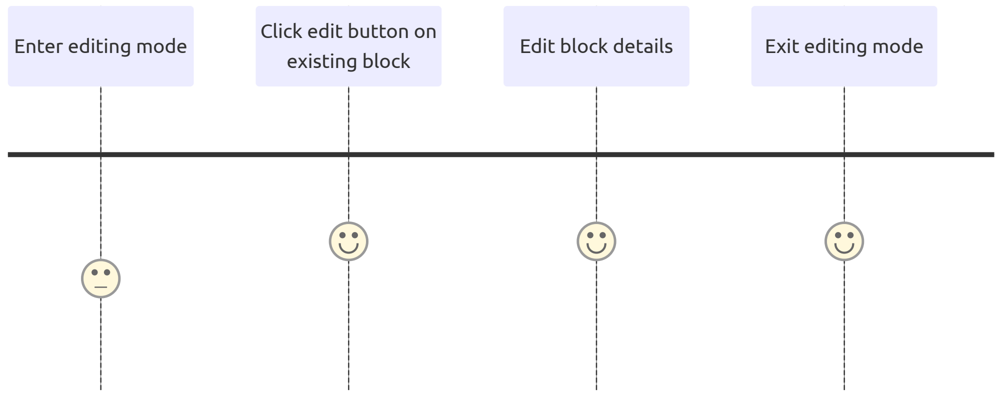


4.5. Profile Page Customization

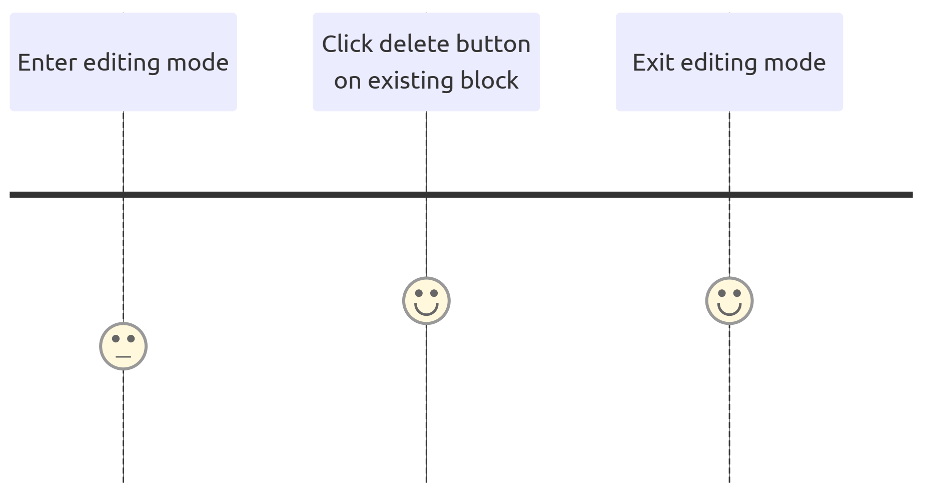
Add Profile Block



Edit Profile Block (Unimplemented)



Delete Profile Block (Unimplemented)



5. Design Process Documentation

We followed a fairly standard Scrum process, and I (Rowan) acted as Scrum master for roughly the latter half of the course.

One process we used that isn't included in the standard Scrum process is writing design docs. After our MVP, for each major feature that we implemented, we wrote a fairly detailed design document that outlined the motivation behind the feature and how the system would work. These design docs included details such as how the user would interact with the feature and what new backend APIs we would need to support the feature.

- 10/6/21: We discussed the problem scenario and user journey for our app. This led us to our initial vision for the app as an anime list that can appeal to a "power user" userbase.
- 10/11/21: After working on the Hello World app, we agreed to use a Next.js / Spring Boot / GraphQL tech stack. We also started brainstorming user stories and creating a set of issues to begin implementation.
- 10/13/21: Agreed to use MongoDB as our database.

- 11/1/21: Completed design document for the user list and custom rating system features. Users will create rating systems that encapsulate all of the information necessary to validate entries in a user list. When creating a new anime list, users will select the rating system and the data for the rating system will be stored as part of the list document.
- 11/4/21: Agreed on the extent to which we want to use TDD. We agreed to try to write unit tests whenever possible and without writing highly redundant test code, and to make limited use of component testing.
- 11/15/21: Completed design document for the profile page customization system. The profile page will consist of a series of blocks, which can each display a given type of content. Users will be able to place and arrange these blocks (semi)-freely.